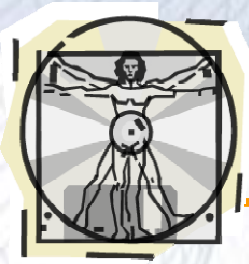
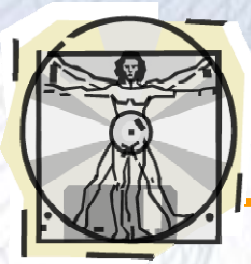
The Tropos Development Methodology



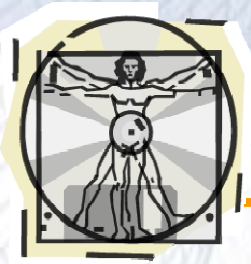
Tropos

- *Tropos* is derived from the Greek *trope* which means “easily changeable” or “easily adaptable.”
- The Tropos Software Methodology was first presented in 2001 at the at 5th IEEE International Symposium on Requirements Engineering where the authors proposed a new specification language, called Formal Tropos, that offers the primitive concepts of early requirements frameworks.



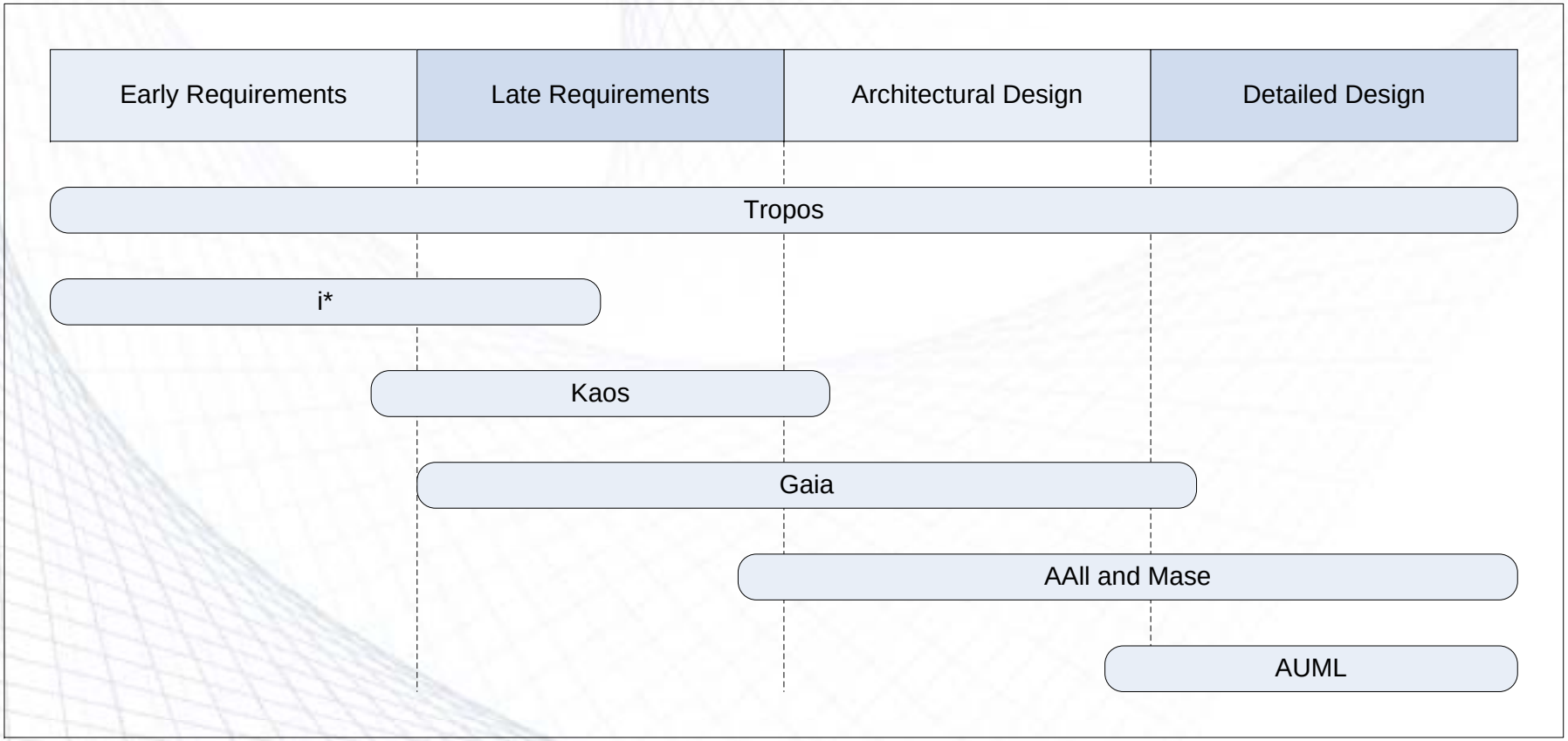
Fundamentals

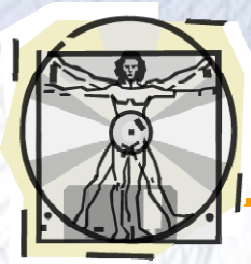
- Tropos has two main features:
  - Agents, goals, plans, and other knowledge level concepts are fundamental primitives in their notation and used informally during the complete software development process.
  - Requirements analysis and specification plays an important role when the system that has to be developed is analyzed with respect to its intended environment.



Comparison

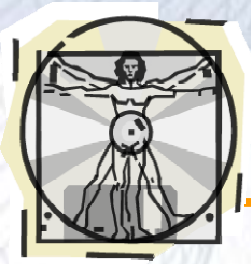
- Tropos is used during the whole software development process.





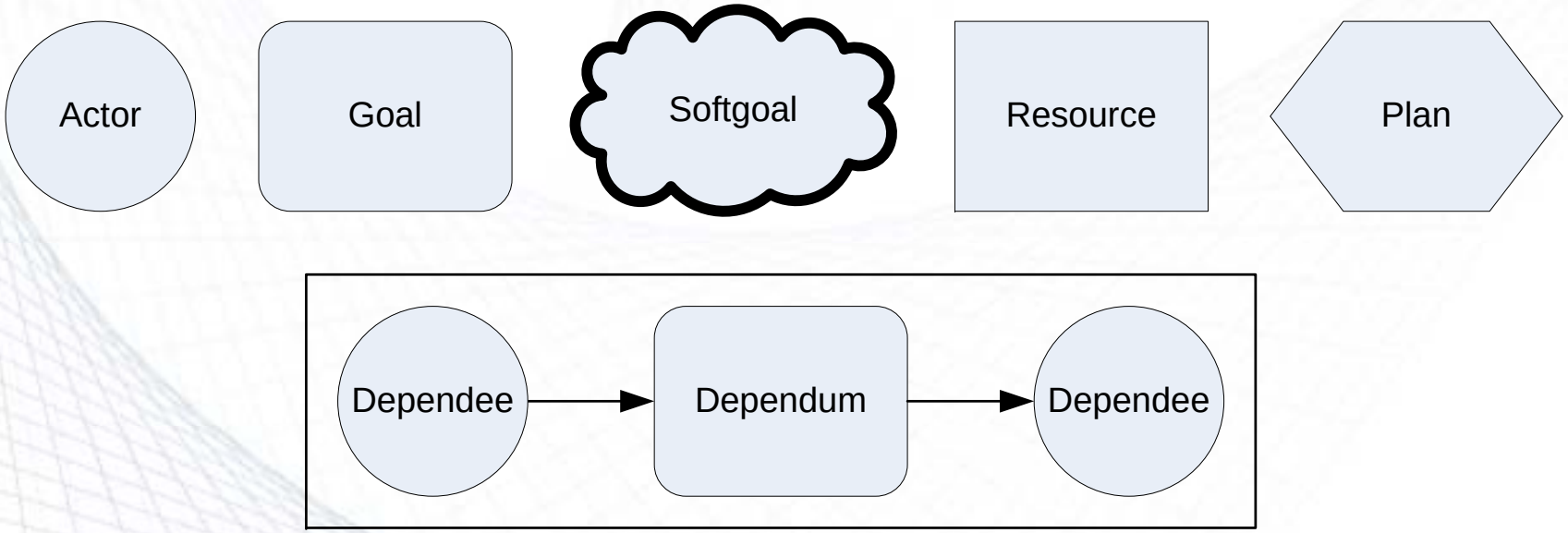
Tropos: Methodology

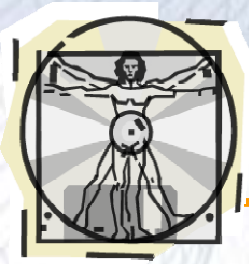
- The phases during software development that are covered by *Tropos* are the following:
  - **Early Requirements.** This phase includes identification of relevant stakeholders (actors) with respective objectives (goals).
  - **Late Requirements.** The target system is introduced as another actor and is related to stakeholder actors in terms of actor dependencies (these indicate the obligations of the system towards its environment).
  - **Architectural Design.** Additional system actors are introduced and assigned to goals or subtasks of system goals or system tasks.
  - **Detailed Design.** All system actors are described in detail (including specifications of communication and coordination); output is a *Tropos* specification.
  - **Implementation.** The *Tropos* specification is transformed to a skeleton for implementation (in detail, this is done by mapping from *Tropos* constructs to those constructs of an agent programming platform; later code is added to the skeleton).



Conceptual Entities /1

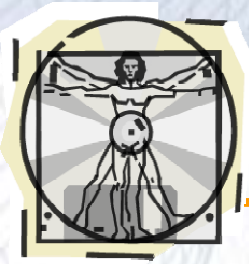
- Symbolic representation of the main concepts of *Tropos*





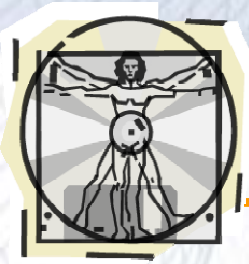
Conceptual Entities /2

- **Actor.** An actor is a generalization of a software agent. This concept models a physical agent or a role (abstract characterization of behavior of an actor within a certain context) or position (set of roles played by one agent).
- **Goal.** This concept is a representation of strategic interests of an actor. *Tropos* makes a difference between goals having no clear-cut definition or criteria (softgoals) and hardgoals.
- **Dependency.** A dependency exists between two actors to indicate that one of the actors (*depender*) depends on the other (*dependee*) to achieve its goals (or to execute some plan, or deliver a resource). The set of a goal, plan, and resource is called *dependum*.
- **Resource.** Physical or informational entities that one actor wants and another can deliver are represented by resources.
- **Plan.** The concept of a plan represents a way of satisfying a goal.
- **Capability.** This means the capability of an actor to define, choose and execute a plan to fulfill a goal in a given operating environment.
- **Belief.** A belief represents the actor's knowledge of the world.



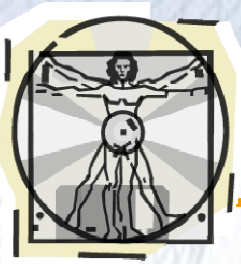
Tropos Models

- Several models can be built using Tropos conceptual entities

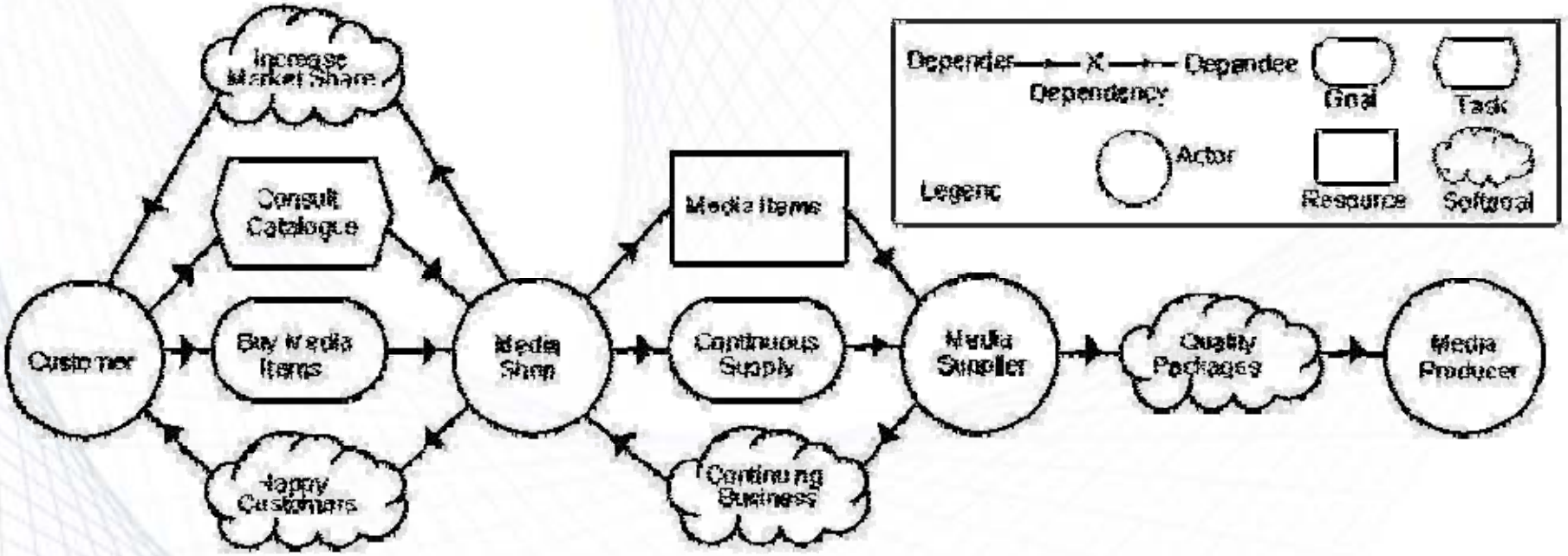


1. Actor & Dependency Model

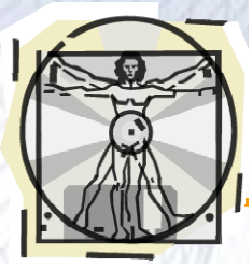
- These types of models result from the analysis of social and system actors and their goals and dependencies for the achievement of goals.
- The models are built in the early requirements phase when application domain stakeholders, their intentions and their inter-dependencies are identified.
- Actor models may be extended during the late requirements phase by adding the future system itself as another actor, along with its inter-dependencies with social actors.
- In architectural design level actor and dependency models provide a more detailed structure of the future system actor and its internal structure.



Sample Actor Diagram

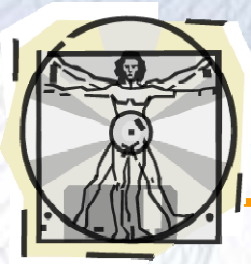


Dependency between two actors: one actor depends on the other to attain some goal, execute some plan or deliver a resource. First actor is called depender, second one is dependee. The object around which the dependency centers is called dependum.



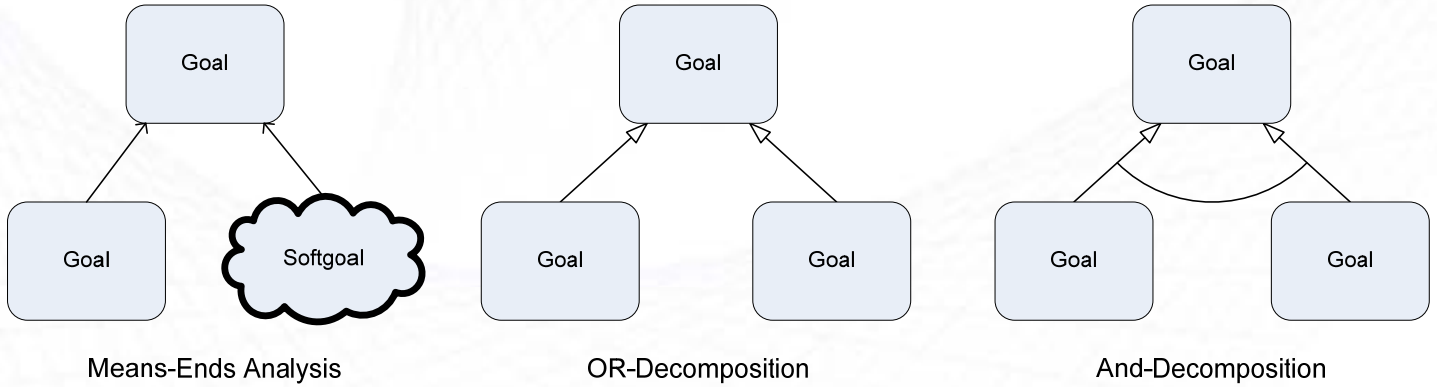
2. Goal and Plan Models

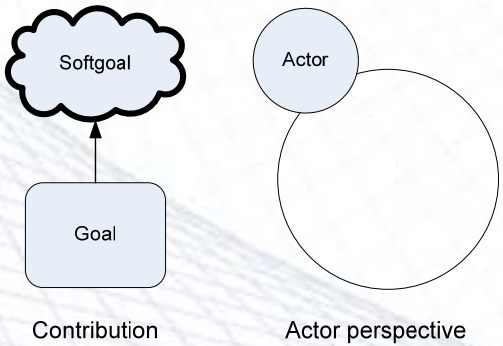
- A designer can analyze goals and plans from an actor's perspective using three techniques:
- **Means-ends analysis.** refining a goal into subgoals to identify plans, resources, and softgoals (which provide means for achieving the goal at the end).
- **AND/OR (de)composition.** allows a combination of AND and OR (de)compositions of a root goal into subgoals (creates a goal structure).
- **Contribution analysis.** to point out goals contributing negatively or positively in reaching the root goal being analyzed. It is a special case of means-end analysis where means are always goals.



Goal and Plan Models

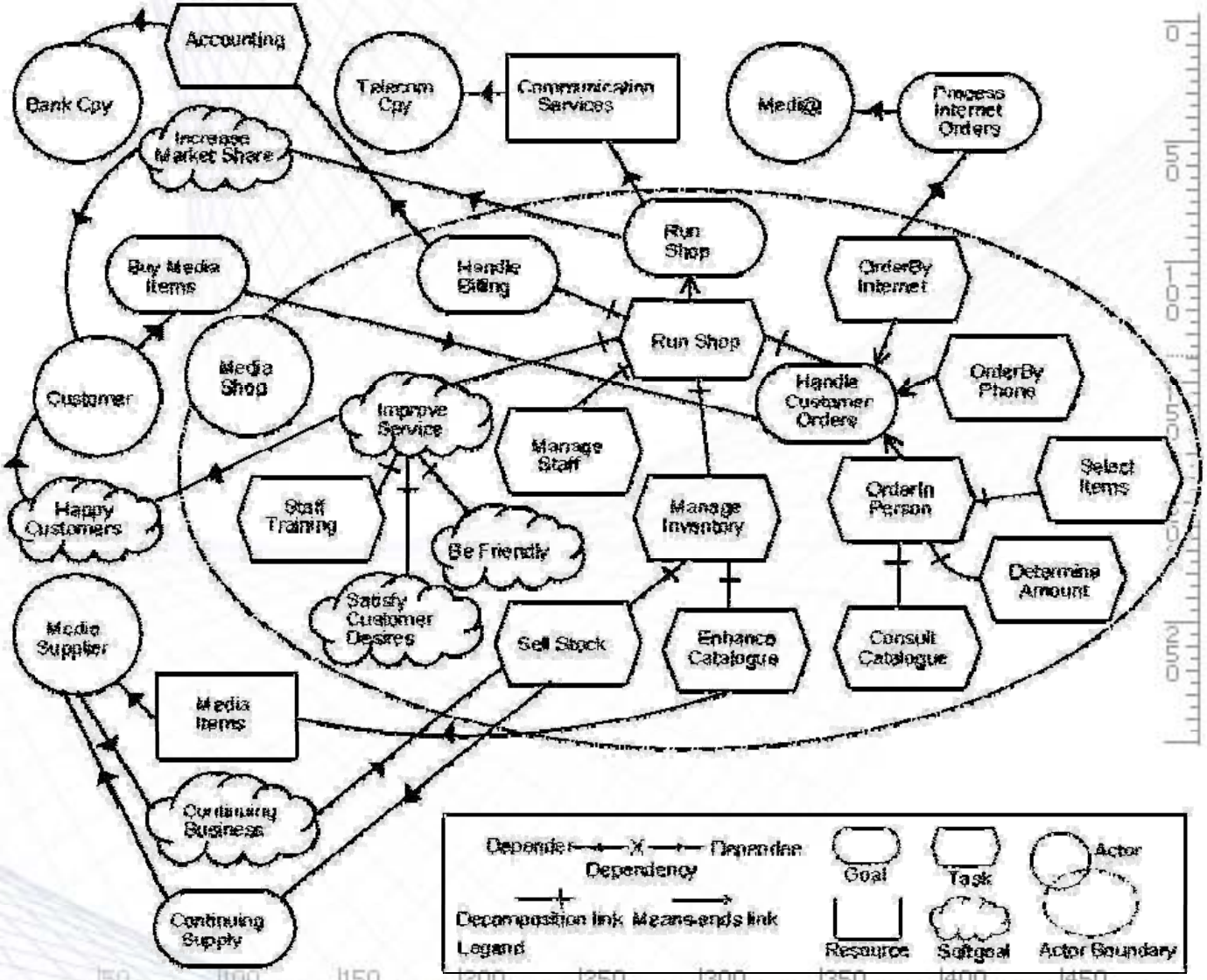
- Samples of *Means-End Analysis*, *OR-Decomposition*, *And-Decompostion*, *Contribution*, and *Actor perspective*.

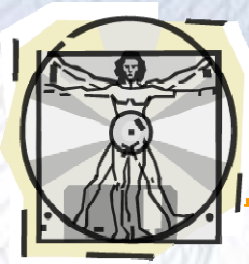




Goal and Plan Models

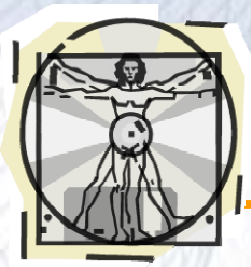
- Sample goal diagram for the actor *MediaShop*





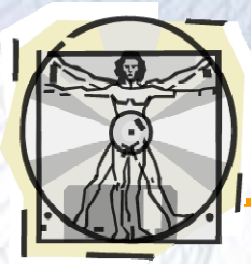
Tropos: Process

- *Tropos* is a generic design process which focuses on a goal analysis for different actors. The process is a non-deterministic concurrent algorithm (including a completeness criterion). During the analysis process the design flows from external to system actors through a goal analysis and delegation.
- The process starts with actors with a list of related goals (and softgoals) and the root goals are analyzed from the actor's perspective. Generated subgoals are delegated to other actors or the actor itself when it deals with the goals itself. This analysis is carried out concurrently with respect to each root goal. The process may require the introduction of new actors which are delegated to goals and/or tasks. The process ends when all goals have been dealt with to the satisfaction of actors.



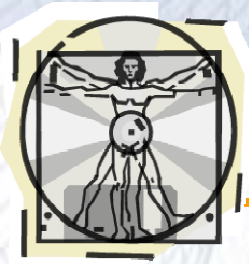
Formal Description of Process

```
global actorList,goalList,agenda,dependencyList,capabilityList,goalGraph;
procedure rootGoalAnalysis(actorList,goalList,goalGraph)
        begin
                rootGoalList = nil;
                for actor in actorList do
                        for rootGoal in goalList(actor) do
                                rootGoalList = add(rootGoal, rootGoalList);
                                rootGoal.actor = actor;
                        end;
                end;
        end;
        concurrent for
                rootGoal in rootGoalList do
                goalAnalysis(rootGoal,actorList)
        end concurrent for;
        if not[satisfied(rootGoalList,goalGraph)] then fail;
end procedure
```



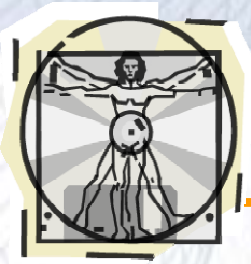
Formal Process: Explanation

actorList	Finite set of actors.
goalList	List of goals for actor (in goalList(actor)).
agenda	List of goals actor has undertaken to achieve personally (initially empty).
dependencyList	List of dependencies between actors.
capabilityList	Includes <goal,plan> pairs indicating means by which the actor can achieve goals.
goalGraph	Stores representation of goal graph that has been generated so far by the design process (initially contains all root goals of all initial actors with no links among them).
rootGoalAnalysis()	Conducts concurrent goal analysis for every root goal.
satisfied()	Checks whether all root goals in goalGraph are satisfied.
goalAnalysis()	Conducts concurrent goal analysis for every sub goal of a given root goal.



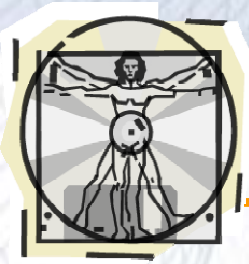
Tropos: Modeling Language

- **Meta-metamodel level.** The *meta-metamodel level* is the basis for the extensions of the metamodel. It contains the language primitives that allow the inclusions of constructs (basic language, structural elements, e.g. attributes, entities).
- **Metamodel level.** The *metamodel level* provides constructs for modeling knowledge level entities and concepts (knowledge level notations, e.g. actor, goal, dependency).
- **Domain model level.** The *domain model level* provides a representation of entities and concepts of specific application domains which are built as instances of the metamodel constructs (application domain entities).
- **Instance Model level.** The *instance model level* contains instances of the domain model.



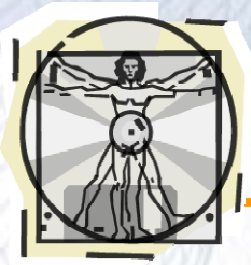
Tropos: Modeling Language

Level	Description	Examples
Meta-Metamodel	Specifies language structural elements	Attribute, Entity
Metamodel	An instance of the meta-metamodel Defines knowledge level notions	Actor, Goal, Plan
Domain	An instance of the metamodel Models application domain entities	
Instance	Instantiates domain model elements	



Design Level

- At the design level several design patterns, such as *Pair* pattern and *Mediation* patterns can be used to implement the system.
  - The *Pair* patterns – such as booking, call-for-proposal, subscription, or bidding – describe direct interactions between negotiating agents.
  - The *Mediation* patterns – such as monitor, broker, matchmaker, mediator, embassy, or wrapper – feature intermediary agents that help other agents to reach an agreement on an exchange of services.
  - Etc.



Conclusions

- Tropos is one of the "better" methodologies in terms of blending agent-based technology and contemporary software engineering practice.